Imperialist Competitive Algorithm for the Flowshop Problem

Gabriela Minetti[1] and Carolina Salto[1,2]

[1] Facultad de Ingeniera, Universidad Nacional de La Pampa
Calle 110 N390, General Pico, La Pampa, Argentina
[2] CONICET
{minettig,saltoc}@ing.unlpam.edu.ar

Abstract. This article presents a new optimization techniques based on Imperialistic Competitive Algorithm to solve the flowshop scheduling problems, which objective is to minimize the makespan. Furthermore, this approach is enhanced by a local search procedure in order to improve the best solutions. To show the efficiency of the proposed methods, we consider many instances of increasing complexity for the flowshop problem. Computational tests are presented and comparisons between the two approaches are made. The analysis of the results suggests that the enhanced approach is capable to find the best solutions of the problem at hand.

Keywords: Imperialist Competitive Algorithm, optimization, flowshop problem

1 Introduction

In many manufacturing and assembly facilities a number of operations have to be completed on every job. Often, these operations must be done on all jobs in the same order, which implies that the jobs have to follow the same route through a set of machines. This problem is known as the Flowshop Sequencing Problem (FSSP) [10], which objective is to determine the sequence of the n jobs so that a certain performance measure will be optimized. The most commonly studied performance measure is the minimization of makespan (c_{max}).

Since it has been proven that the FSSP with makespan minimization is NP-complete when the number of machines is greater or equal than three [8], there have been proposed exact, (constructive and improvement) heuristic, and (trajectory and population-based) metaheuristic methods. The exact methods need exponential computation time in most cases, as a result they are feasible for only small size problems, as shown in [12, 13, 21]. On the one hand, many constructive heuristic methods have been introduce since 1954 in [6, 10, 11, 15, 17]. On the other hand, the improvement heuristics enhance an existing initial FSSP solution using a local search method [5, 22]. Concerning the metaheuristics for FSSP, some research are based on trajectory methods, as we can see in [7, 16,

20]. Contrary to trajectory methods, population-based metaheuristics have been proposed to solve FSSP in [4, 18, 19, 23].

The Imperialist Competitive Algorithm (ICA) [1] is a new population-based metaheuristic that is inspired by the imperialistic competition, where the countries (solutions to the problem at hand) are divided into two types: imperialist and colonies. The base of the algorithm is the imperialistic competition, where the colonies move toward the empires, empires compete among them and the weak ones collapse. ICA has been successfully used to solve several NP-Hard optimization problems, e.g. non-convex dynamic economic power dispatch [3], constrained economic dispatch [14], complicated image matching [9], among others. The objective of this work is to research the potential of this metaheuristic to solve the FSSP in an efficient way. By doing this, we propose the ICA_{FSSP} algorithm to solve problems with solution represented by permutations. Consequently, we analyse their performance using different operators to produce the movements of the colonies toward the imperialistic countries. Also, we present an improved ICA_{FSSP} called ImpICA_{FSSP}, which implements an improvement procedure in order to simulate the growth of the imperialistic countries. We analyze the behavior of both ICAs over a wide range of complex FSSP instances.

The rest of this article is organized as follows. Section 2 describes in detail the flowshop problem. In Section 3, we describe the metaheuristic ICA to solve the FSSP (ICA_{FSSP}). In Section 4, we outline an improved version of the ICA_{FSSP}, called ImpICA_{FSSP}. Section 5 explains the experimental design. Then, we study and analyze the results obtained by ICA_{FSSP} and ImpICA_{FSSP} for FSSP in Section 6. Finally, we present our principal conclusions and future lines of research in Section 7.

2 FSSP Description

The flowshop scheduling problem is generally described as follows: each job from the job set $J = \{1, 2...n\}$, for $n > 1$, has to be processed on m machines in the order given by the indexing of the machines $(1, 2..., m)$. Thus job j, $j \in J$, consists of a sequence of m operations; each of them corresponds to the processing of job j on machine i for an uninterrupted processing time $t_{ji} \geq 0$. It is assumed that a zero processing time on a machine corresponds to a job performed by that machine in an infinitesimal time. Machine i, $i = 1, 2...m$, can execute at most one job at a time, and it is assumed that each machine processes the jobs in the same order. The objective is to find a sequence of jobs that minimizes the maximum flow time, which is called *makespan* (c_{max}).

If we let $c(j_j, i)$ denote the completion time of job j_j on machine i and let $\{j_1, j_2, ..., j_n\}$ denote a job permutation of J, then we can calculate the completion times for a n jobs and m machines flowshop problem as shown in Equation 1, where the final makespan is $c_{max} = c(j_n, m)$.

$$\begin{array}{ll} c(j_1, 1) = t_{j_1 1}, & \\ c(j_1, i) = c(j_1, i-1) + t_{j_1 i}, & i = \{2, ..., m\} \\ c(j_j, 1) = c(j_{j-1}, 1) + t_{j_j 1}, & j = \{2, ..., n\} \\ c(j_j, i) = \max\{c(j_{j-1}, i), c(j_j, i-1)\} + t_{j_j i}, & j = \{2, ..., n\}, i = \{2, ..., m\} \end{array} \quad (1)$$

As we can see, this kind of problem is essentially a permutation schedule problem, and the permutation of jobs can be naturally represented by a sequence of positive integers. For example, $S = [2, 3, 1, 4]$ means that the corresponding job sequence is j_2, j_3, j_1, and j_4.

3 Imperialist Competitive Algorithm for the FSSP

The ICA was first proposed in [1], inspired by the imperialistic competition. It is a population-based metaheuristic for solving optimization problems. It starts with an initial population of countries (solutions), which are categorized into two groups: imperialist (the best solutions) and colonies (the remaining solutions). Then, the algorithm iterates over a set of steps to improve the population, producing imperialistic competitions among these empires. Powerful imperialists try to increase in the power and absorb colonies to their empire from weak imperialists, whose power gradually decreases. When an empire loses all of its colonies, this empire collapses. The competition ends when only one empire exists, and the best solution of the problem is represented by its imperialistic country.

In this article, we propose an ICA to solve a permutation-based problem like the flowshop problem, denominated ICA_{FSSP}. The flowchart of this algorithm is presented in Figure 1. The steps of the ICA_{FSSP} are described as follows:

Step 1: Initialize all countries. An initial size set, N_C, of countries is created by a random procedure.

Step 2: Determine the imperialistic countries. After calculating the fitness function to each country $c_i = 1/c_{max}$, the best N_{imp} of them are selected as imperialists. The rest $N_{col} = N_C - N_{imp}$ countries are the colonies.

Step 3: Assign the colonies to each imperialist according to ist power. The colonies are divided among imperialists based on their power, in order to form the initial empires. To calculate the power p_j of each imperialistic country, first a normalized power of each one is calculated according to Equation 2.

$$p_j = \frac{c_j}{\sum_{i=1}^{N_{imp}} c_i} \tag{2}$$

The number of colonies of each imperialistic country (N_{col_j}) is proportional to its power, and it is determined by Equation 3 [3]. In order to assign colonies to an imperialist j, N_{col_j} colonies are randomly selected between the remaining ones.

$$N_{col_j} = p_j \times N_{col} \tag{3}$$

Step 4: Move the colonies toward their relevant imperialist. The colonies in each of the empires start moving toward their relevant imperialistic country. In this work, as the solutions are represented by permutations, those movements are accomplished by crossover and mutation operators. The crossover operator is applied in such a way that a colony and its imperialist country are considered as parents. The most popular and standard

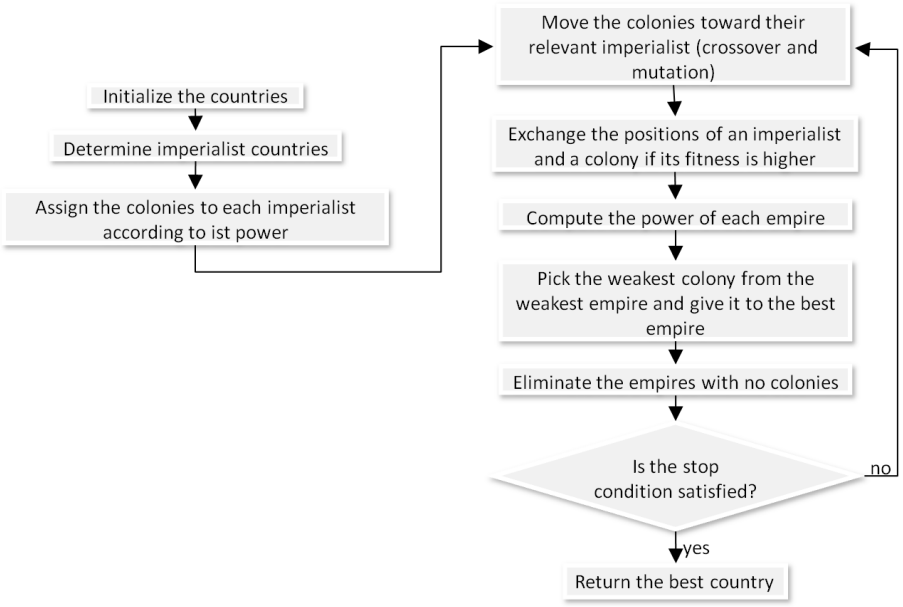

Fig. 1: The flowchart of the ICA_{FSSP}.

permutation-based crossover operators are the PMX and the OX. Regarding mutation operators, they are applied to each colony in order to simulate a randomly deviated direction, as the original proposal of ICA. In particular, we can use insert, pairwise interchange and inversion operator for permutation-based representation.

Step 5: Exchange the positions between the imperialist and colony if the colony fitness is higher than imperialistic one. When moving toward the imperialistic country, a colony might reach a position with higher fitness than of its related imperialist country. In this case, the imperialist and the colony should change their positions. If there are several colonies better than the imperialistic country, then the best colony will replace the imperialist.

Step 6: Compute the total power of each empire. The total power of each empire (tp_j) is computed including the power of the imperialistic country and the power of its colonies, according to the Equation 4 [1]. In this equation appears the parameter $\sigma \in [0, 1]$, which causes that the total power of the empire might be determined by just the imperialistic country ($\sigma = 0$) or by the colonies increasing the σ value. We adopt the value of $\sigma = 0.1$ in our implementation as suggested in [1].

$$tp_j = c_j + \sigma \times \sum_{i=1}^{N_{col_j}} c_i \tag{4}$$

Step 7: Pick the weakest colony from the weakest empire and give it to the best empire. Select the empire with the highest tp_j as the best one and increase its number of colonies by one ($N_{col_j} + 1$). The colony of the weakest empire, k, with the lowest c_i is considered as the weakest one. Decrease the number of colonies of the weakest empire by one ($N_{col_k} - 1$).

Step 8: Eliminate the empires with no colonies. The number of empires decreases by one ($N_{imp} - 1$) if the weakest empire j has $N_{col_k} = 0$.

Step 9: Stop condition. If more than one empire remained ($N_{imp} > 1$) and the $time < time_{max}$, then go to Step 4.

Step 10: End. The best imperialistic country is returned as the best solution to the optimization problem.

Finally, we remark that ICA algorithms only need two parameters: the population size (N_C) and the empire number (N_{imp}). As a consequence, the time incurred in fine-tunning the parametric values is drastically reduced.

4 Improvements to the ICA$_{FSSP}$

The ICA$_{FSSP}$ for the FSSP, discussed above, is very simple and easy to implement. However, it can become inefficient because no improvements are applied to the imperialistic country, i.e. the imperialist solution, does not undergo to any improvement procedure, while their colonies move to it (Step 4) and no one reaches a better position (Step 5).

Consequently, we suggest a modification to the ICA$_{FSSP}$ to overcome the problem mentioned in the previous paragraph. The resultant algorithm is named ImpICA$_{FSSP}$, which incorporates a local search procedure to improve the imperialistic country. Introducing this improvement step (after Step 4) the imperialists try to become more powerful and continue with their dominance. The local search procedure applies repeatedly the interchange operator on the imperialistic country and works as follows: the first job in the first position of the imperialist solution x, denominated x_1, is interchanged for a job in a different randomly selected position. If this new solution x' has a higher fitness value, the current imperialistic solution is replaced by the new x' and the local search stops. If no improvement is found, the procedure continues with the job in the next position x_2. The search iterates through all the jobs x_i, without repetition. The basis of this procedure is to make a very quick local search.

5 Experimental Design

In this section, we describe the experimental design used in this approach and the execution environment. In order to evaluate the proposed algorithms ICA$_{FSSP}$ and ImpICA$_{FSSP}$ to solve the FSSP, we have selected a wide range of instances used in the literature taking into a account the complexity of them. This complexity is given by the number of jobs and machines. The Taillard's instances with 20, 50, 100, 200, and 500 jobs and 10 and 20 machines were chosen, whose

best known values are available in the OR Library [2]. In this benchmark set, 10 instances for each problem size are provided totalizing 90 instances.

Due to the stochastic nature of the algorithms, we performed 30 independent runs of each trial to gather meaningful experimental data and to apply statistical confidence metrics to validate our results and conclusions. Consequently, a total of 2700 executions (90 instances $\times$ 30 independent runs) were carried out by each algorithm. The hardware involved in the experimentation was an INTEL I7 3770K, 8 GB RAM, and the Slackware Linux with 2.6.27 kernel version.

To evaluate our algorithms, we use as performance measure the relative difference (*gap*) between the cost of the best solution found by the algorithm ($BestCost$) in each trial and the best known cost ($BestKnownCost$) for each instance . The *gap* is calculated as shown in Equation 5.

$$gap = \frac{Best\,Cost - Best\,Known\,Cost}{Best\,Known\,Cost} \tag{5}$$

The parameters that our ICA algorithms use are the population size, the number of empires, and the complementary stop condition. The size of the population (N_C) of a metaheuristic is crucial to the performance of the algorithm and as the ICA is not the exception, two different values of N_C are considered: 50 and 100 (these values are widely used in the literature). After an experimentation on all instances, the *gap* values found by ICA$_{FSSP}$ with N_C=100 are less than the ones of case of N_C=50. The Kruskal-Wallis test confirms this situation, indicating that there are statistically significant differences between groups of ranks (p-values higher than 0.01, the significance level). For that reason, N_C=100 is the country size used in the remaining experimentation. The value of N_{imp} is set to the 10% of N_C because it is a typical set in the literature. The following criteria are used to determine the stop condition: the existence of an only one empire or to reach the maximum time $time_{max}$ of execution fixed in $n \times n/2 \times 30ms$ [19].

6 Result Analyisis

An analysis of the results obtained by ICA_{FSSP} and ImpICA_{FSSP} is presented in this section. First, we compare the ICA_{FSSP}'s behavior considering different combinations of crossover and mutation operators. Finally, we study the performance of ICA_{FSSP} and ImpICA_{FSSP} algorithms.

6.1 Analysis of ICA_{FSSP} behavior

In this section, we investigate the impact of different combinations of crossover and mutation operators on the ICA_{FSSP} search. Recall that these operators are applied in the process of movement of the colonies toward their imperialistic country. Two well known permutation-based crossover operators, such as OX and PMX, and three mutation operators, insertion (ins), interchange (int), and

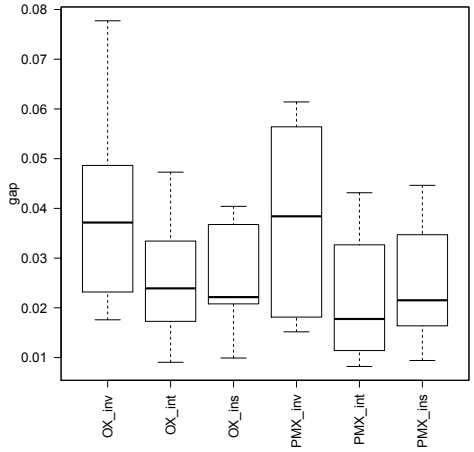

Fig. 2: Boxplot of the *gap* values for ICA_{FSSP} and each combination of operators

Table 1: Duncan test

Groups	Combinations
a	OX_inv
a	PMX_inv
b	OX_int
b	PMX_ins
c	OX_ins
c	PMX_int

inversion (inv) are considered in this analysis. Consequently, a total of six different combinations of these operators are obtained (for example, OX_inv means the use of OX as crossover operator and inversion as mutation operator). ICA_{FSSP} with each combination of operators was run for the whole set of instances.

The first considered quality indicator is the *gap* value. Figure 2 provides a revealing summary of this indicator by means of a boxplot considering the gaps values obtained for the total of the instances. In general, the combinations with interchange mutation give the best *gap* values, followed by insertion. The boxes with the inversion mutation are much higher than the remaining ones, suggesting differences between groups. Furthermore, we observe that the best combination is the PMX_int, whose gap values are between 0.013 and 0.032. Moreover, the minimum *gap* value (0.008) is obtained for ICA_{FSSP} with the PMX_int combination, followed by the OX_ins (0.009 *gap* value).

Given that the p-value of the Friedman test is less than the significance level ($\alpha = 0.01$), we can conclude that significant differences arise between the observed results. Attending to this result, a post-hoc statistical analysis help us to detect concrete differences between algorithms. For this purpose, we use the Duncan's Multiple Range Test and present the Table 1 for the duncan's output, where combinations with the same letter are not significantly different. In average, the worst results are obtained by the two combinations with the inversion operator (group a). In contrast, the minimum average gap values are found by the OX_ins and PMX_int combinations belonging to the group c. Consequently, the best combination of operators is given by PMX and interchange, as crossover and mutation operator, respectively.

Another interesting measure for the ICA_{FSSP} with the different operator combinations is the runtime to find the best solution, as shown in Table 2. In

Table 2: Average of the ICA_{FSSP} runtime values to find the best solution considering different operators and for each problem size.

Instance	OX			PMX		
	Inv.	**Int.**	**Ins.**	**Inv.**	**Int.**	**Ins.**
20×10	0.07	0.08	**0.05**	0.07	0.09	**0.05**
20×20	0.16	0.12	**0.08**	0.15	0.11	0.11
50×10	1.29	0.78	0.65	1.01	**0.64**	1.03
50×20	**1.03**	1.83	1.43	1.55	1.34	1.87
100×10	5.74	5.81	**1.93**	7.45	5.31	6.15
100×20	13.52	12.28	**8.07**	11.39	10.29	14.54
200×10	37.26	30.29	31.56	37.17	**19.85**	21.30
200×20	76.78	78.04	75.94	78.62	**51.28**	89.96
500×20	907.84	900.65	565.02	801.05	**540.60**	735.88
Average	115.96	114.43	76.08	104.27	**69.95**	96.76

this case, the mean runtime in seconds is displayed. In average, ICA_{FSSP} with PMX_int is the fastest approach.

Summarizing, we can conclude that ICA_{FSSP} with PMX_int is the best approach because the best *gap* values, meaning best schedules, are obtained in the minimum execution time. Therefore, PMX operator and interchange mutation are the operators selected to be used in the following experimentation.

6.2 Comparison between ICA$_{FSSP}$ and ImpICA$_{FSSP}$

In this section, we compare the behaviour of ICA$_{FSSP}$ and its improved version ImpICA$_{FSSP}$ on the FSSP, see Table 3. Figures in the two first columns stand for the average of the minimum best *gap* values (column Gap$_{Best}$) and the mean best *gap* values (column Gap$_{Mean}$). While, figures in the last column present the average number runtime (in seconds) to reach the best value (Time$_{Best}$). The minimum values of Gap$_{Mean}$ column are bolded and the symbol "$*$" indicates that the respective p-values for t-test are less than the 0.01 significance level.

From the results of Table 3, we can observe that the mean best *gap* obtained by ImpICA$_{FSSP}$ is less than the found by ICA$_{FSSP}$, for the all groups of instances. This indicates that the solution quality given by ImpICA$_{FSSP}$ is higher than the obtained ones by ICA$_{FSSP}$, suggesting the superiority of our improved ICA. This shows nicely how the effectiveness of adding an improving phase to the imperialistic countries contributes positively to the search process. Furthermore, we can notice that statistical significant differences exist between these options (p-values $< 4.2e^{-09}$). Considering the spent time to find the best solution, ImpICA$_{FSSP}$ is faster than ICA$_{FSSP}$ when the most complex instances are solved. Therefore, we can conclude that ImpICA$_{FSSP}$ produces more optimal schedules than the ICA$_{FSSP}$ and is more efficient when the complexity grows.

7 Conclusions

In this paper, we presented ICA$_{FSSP}$ and ImpICA$_{FSSP}$, two new metaheuristic approaches based in the Imperialist Competitive Algorithm for solving the flow-

Table 3: Best and average *gap* values and runtime to find the best solution for ImpICA$_{FSSP}$ and ICA$_{FSSP}$.

Instance	**ImpICA$_{FSSP}$**			**ICA$_{FSSP}$**		
	Gap_{Best}	Gap_{Mean}	$Time_{Best}$	Gap_{Best}	Gap_{Mean}	$Time_{Best}$
20×10	0.010	**0.041***	0.39	0.011	0.052	0.09
20×20	0.010	**0.035***	0.78	0.014	0.044	0.11
50×10	0.017	**0.054***	5.33	0.042	0.083	0.64
50×20	0.038	**0.069***	6.04	0.043	0.100	1.34
100×10	0.011	**0.037***	18.66	0.010	0.055	5.31
100×20	0.032	**0.067***	15.72	0.033	0.097	10.29
200×10	0.008	**0.027***	55.83	0.008	0.049	19.85
200×20	0.026	**0.068***	39.10	0.026	0.098	51.28
500×20	0.016	**0.054***	193.12	0.018	0.072	540.60
Average	0.019	**0.050**	37.29	0.023	0.072	69.95

shop sequencing problem. ICA$_{FSSP}$ is the result of adapting the original ICA to solve a permutation-based problem like FSSP, which uses crossover and mutation operators based on this codification to move the colonies toward the imperialistic countries. The results indicates that ICA$_{FSSP}$ with the combination of PMX and interchange operators obtains the best quality solutions.

Our second proposal is ImpICA$_{FSSP}$, an improved version of ICA$_{FSSP}$ considering the application of a local search procedure to the imperialistic countries in order to increase its power. This improvement allows to obtain better schedules than the found by ICA$_{FSSP}$ in all tested instances.

As further research lines, we would first attempt to fine-tune the algorithmic design with the aim of improving the current results for the FSSP. As a second research line, we plan to adapt this metaheuristic to solve related problems.

Acknowledgements

The authors are supported by the Universidad Nacional de La Pampa, and the ANPCYT under contract PICTO 2011-0278 and the Incentive Program from MINCyT. The second author is also supported by CONICET.

References

1. E. Atashpaz-Gargari and C. Lucas. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4661–4667, Sept 2007.
2. J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.
3. M. Behnam, R. Abbas, S. Alireza, and Mehdi E. Imperialist competitive algorithm for solving non-convex dynamic economic power dispatch. *Energy*, 44(1):228 – 240, 2012.
4. P. Chang, W. Huang, J. Wu, and T.C.E. Cheng. A block mining and recombination enhanced genetic algorithm for the permutation flowshop scheduling problem. *International Journal of Production Economics*, 141(1):45–55, 2013.

5. J. P. Fan and G. K. Winley. A heuristic search algorithm for flow-shop scheduling. *Informatica: journal of computing and informatics*, 32(4):453–464, 2008.
6. J. M. Framinan and R. Leisten. An efficient constructive heuristic for flowtime minimisation in permutation flowshops. *Omega*, 31(4):311317, 2003.
7. J. Gao, R. Chen, and W. Deng. An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 51(3):641–651, 2013.
8. M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.
9. L. Huang, H. Duan, and Y. Wang. Hybrid bio-inspired lateral inhibition and imperialist competitive algorithm for complicated image matching. *Optik - International Journal for Light and Electron Optics*, 125(1):414 – 418, 2014.
10. S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1:61–68, 1954.
11. C. Koulamas. A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105(1):66–71, 1998.
12. Z. A. Lomnicki. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. *OR*, 16(1):pp. 89–100, 1965.
13. G. B. McMahon and P. G. Burton. Flowshop scheduling with the branch and bound method. *Operations Research*, 15(3):473481, 1967.
14. G. Mokhtari, A. J. Ghanizadeh, and Ebrahimi E. Application of imperialist competitive algorithm to solve constrained economic dispatch. *International Journal on Electrical Engineering and Informatics*, 4(4):553–562, 2012.
15. M. Nawaz, E. Enscore Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91 – 95, 1983.
16. V. B. Nouri, P. Fattahi, and R. Ramezanian. Hybrid firefly-simulated annealing algorithm for the flow shop problem with learning effects and flexible maintenance activities. *International Journal of Production Research*, 51(12):3501–3515, 2013.
17. D. Palmer. Sequencing jobs through a multi-stage process in the minimum total time-A quick method of obtaining a near optimum. *Operational Research*, 16(1):101–107, 1965.
18. C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computing Operation Research*, 22:5–13, 1995.
19. R. Ruiz, C. Maroto, and J. Alcaraz. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega*, 34(5):461–476, 2006.
20. R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.
21. E. F. Stafford. On the development of a mixed-integer linear programming model for the flowshop sequencing problem. *The Journal of the Operational Research Society*, 39(12):pp. 1163–1174, 1988.
22. S. M. A. Suliman. A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics*, 64(1-3):143–152, 2000.
23. X. Wang and L. Tang. A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. *Applied Soft Computing*, 12(2):652 – 662, 2012.